Práctica 4: Módulos Odoo Modelo y Vista

Eder Martínez Castro

13 de diciembre de 2025

Índice general

1. Introducción 2
2. Actividad 01 - Lista de tareas 3
   - 2.1. Objetivo 3
   - 2.2. Agregar vista Kanban 3
   - 2.3. Comprobación de la vista Kanban 5
   - 2.4. Agregar vista Calendar 5
   - 2.5. Comprobación de la vista Calendar 10
3. Actividad 02 - Biblioteca de cómics 11
   - 3.1. Objetivo 11
   - 3.2. Modelo Socios 11
   - 3.3. Modelo Ejemplares 14
   - 3.4. Comprobación de su funcionamiento 18
4. Actividad 03 - Pacientes y médicos 21
5. Actividad 04 - Ciclos formativos 21
6. Conclusión 21

1. Introducción

El objetivo de esta práctica es seguir trabajando con el modelo y la vista de los módulos de Odoo, para ello tendremos que implementar cuatro módulos, **Lista de tareas**, **Biblioteca de cómics**, **Pacientes y médicos**, **Ciclos formativos**.

Antes de proceder a empezar con la práctica se nos recomienda revisar los ejemplos del 01 al 06 que tenemos en el siguiente repositorio github.com/sergarb1/OdooModulosEjemplos.git.

Para ello deberemos clonarlo en nuetro equipo utilizando el siguiete comando:

```
git clone https://github.com/sergarb1/OdooModulosEjemplos.git
```

Cuando ya lo tengamos clonado lo que haremos es copiar los 6 ejemplos y pegarlos en nuestro proyecto para poder probarlos y revisarlos. Y nos debería quedar nuestra carpeta con nuestro módulos tal que así:

Figura 1: Carpeta addons, que contiene nuestros módulos de Odoo donde podemos ver los módulos de ejemplo del repositorio para probarlos.

Entender estos módulos de ejemplo nos ayudará mucho para poder hacer la actividad 1 de lista de tareas y la 2 de la biblioteca de comics.

2. Actividad 01 - Lista de tareas

2.1. Objetivo

Para esta actividad se nos pide coger el código de nuestro módulo de ejemplo 2 de la lista de tareas, para modificarlo e implementar en el que podamos verlo además del **modo lista**, poder verlo en **modo kanban** y **calendario**.

2.2. Agregar vista Kanban

Para poder implementar la vista estilo kanban solo necesitamos modificar el archivo `views.xml` que es nuestra vista para poder implementar este nuevo modo de visualización.

```
<!-- ================================================
     VISTA DE KANBAN
     ================================================ -->
<record id="view_lista_tareas_kanban" model="ir.ui.view">
  <!-- Nombre interno de la vista -->
  <field name="name">lista.tareas.kanban</field>
  <!-- Modelo al que pertenece esta vista -->
  <field name="model">lista_tareas.lista</field>
  <!-- Estructura XML de la vista -->
  <field name="arch" type="xml">
    <!-- Vista tipo kanban -->
    <kanban default_order_by="realizada">
      <field name="tarea"/>
      <field name="prioridad"/>
      <field name="urgente"/>
      <field name="realizada"/>

      <templates>
        <!-- Tarjeta de las tareas de kanban -->
        <t t-name="kanban-box">
          <div class="oe_kanban_card oe_kanban_global_click">
            <strong><field name="tarea"/></strong>
            <div>
              <div>Prioridad: <field name="prioridad"/></div>
              <div>Urgente: <field name="urgente"/></div>
              <div>Realizada: <field name="realizada"/></div>
            </div>
          </div>
        </t>
      </templates>
    </kanban>
  </field>
</record>
```

Figura 2: Nuevo fragmento de código para mostrar las tareas en formato kanban.

En este fragmento de código implementamos la nueva vista tipo kanban, que nos mostrará nuestras tareas en tarjetas individuales con los datos de la tarea.

Estrucutura principal de la vista Kanban:

- `<kanban>`: muestra la vista en formato kanban como bien indica el nombre
- `<field>`: declaramos los campos que tendra la vista.
- `<templates>`: contiene la plantilla `kanban-box` donde creamos la tarjeta de la tarea del kanban:
  - La tarjeta es interactiva al usar `oe_kanban_global_click` si hacemos click en ella nos abrira el formulario para poder editarla.
  - Se muestran los datos de la tarea: nombre, prioridad, urgencia y si está o no realizada.

```
<!-- ================================================
     ACCIÓN PRINCIPAL DEL MÓDULO
     ================================================
     Esta acción es la que se ejecutará cuando el usuario
     entre en el menú "Ver tareas".
     Abre el modelo lista_tareas.lista mostrando sus vistas.
-->
<record model="ir.actions.act_window" id="action_lista_tareas">
  <!-- Título visible de la ventana -->
  <field name="name">Listado de tareas</field>

  <!-- Modelo al que se refiere la acción -->
  <field name="res_model">lista_tareas.lista</field>

  <!-- Tipo de vistas que se mostrarán:
       list → vista en tabla
       form → vista en formulario
       kanban -> vista kanban -->
  <field name="view_mode">list,form,kanban</field>
</record>
```

Figura 3: En el bloque para las acciones de nuestro módulo debemos agregar como se ve en pantalla en el `view_mode` el nuevo tipo kanban.

En esta modificación lo que hicimos fue agregar la vista kanban al `view_mode`, porque Odoo utiliza este parámetro para determinar que vistas estarán disponibles en la interfaz de nuestro módulo.

2.3. Comprobación de la vista Kanban

A continuación vamos a enseñar a través de capturas de pantalla el correcto funcionamiento de nuestra vista kanban.

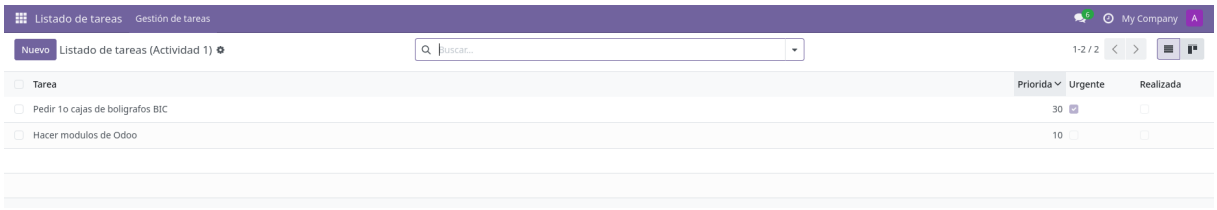


Figura 4: En esta captura podemos ver la vista modo lista de nuestro módulo de listas de tareas que es la primera que nos saldrá cuando lo abramos.

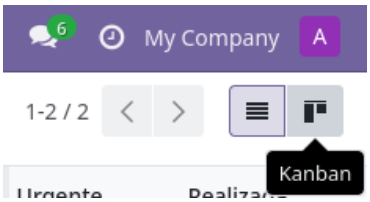


Figura 5: Como se ve en esta captura para cambiar de vista nos iremos a estos iconos y pulsaremos en el que pone kanban para poder ver nuestras tareas en este formato

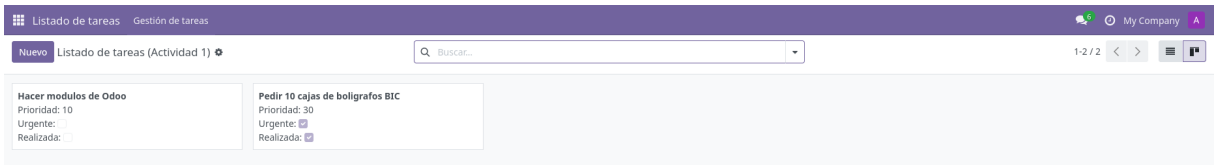


Figura 6: En esta captura podemos ver la vista modo kanban de nuestro módulo de listas de tareas.

Aquí podemos ver nuestras tareas en tarjetas con todos sus campos relevantes, lo que muestra que tenemos la vista del kanban bien implementada.

2.4. Agregar vista Calendar

Para poder implementar la vista estilo calendar deberemos modificar el archivo `views.xml` que es nuestra vista y el archivo `models.py` que es nuestro modelo para poder implementar este nuevo modo de visualización.

Primero vamos a empezar modificando nuestro `models.py`:

```
# Campos para la fecha de inicio y fin de la tarea
fecha_inicio = fields.Date(string="Fecha inicio")
fecha_fin = fields.Date(string="Fecha fin")
```

Figura 7: En esta captura podemos ver como hemos agregado a nuestra modelo los campos `fecha_inicio` y `fecha_fin` para poder marcar el comienzo de una tarea y su final.

Para que nuestra vista calendario pueda mostrar nuestras tareas, necesitamos crear dos nuevos campos para poder marcar la fecha de inicio de la tarea y la fecha de fin de esta.

- `fecha_inicio`: fecha en la que esta tarea empieza.
- `fecha_fin`: fecha límite de la tarea.

Ahora vamos a realizar las siguientes modificaciones en nuestro `views.xml`:

```
<!-- ====================================================
     VISTA DE CALENDARIO
     ==================================================== -->
<record id="view_lista_tareas_calendar" model="ir.ui.view">
  <!-- Nombre interno de la vista -->
  <field name="name">lista.tareas.calendar</field>
  <!-- Modelo al que pertenece esta vista -->
  <field name="model">lista_tareas.lista</field>
  <!-- Estructura XML de la vista -->
  <field name="arch" type="xml">
    <!-- Vista tipo calendar -->
    <calendar date_start="fecha_inicio" date_stop="fecha_fin">
      <field name="tarea"/>
      <field name="prioridad"/>
      <field name="urgente"/>
      <field name="realizada"/>
    </calendar>
  </field>
</record>
```

Figura 8: Nuevo fragmento de código para mostrar las tareas en formato calendar.

En este fragmento de código implementamos la nueva vista tipo calendar, que nos mostrará un calendario con nuestras tareas desde el dia que empiezan hasta el dia que deben estar finalizadas.

Estrucutura principal de la vista calendar:

- `<calendar>`: muestra la vista en formato calendario como bien indica el nombre, esta recibe dos atributos:
  - `date_start="fecha_inicio"`: indicamos el día de comienzo de la tarea.
  - `date_start="fecha_fin"`: indicamos el día final de la tarea.
- `<field>`: declaramos los campos que tendra la vista.

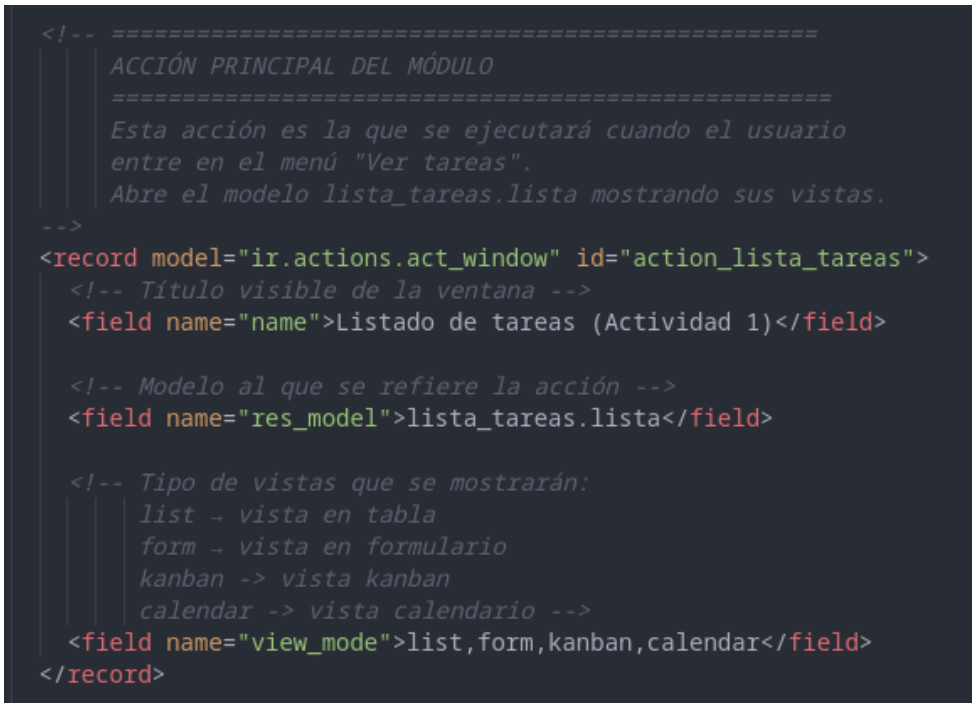


Figura 9: En el bloque para las acciones de nuestro módulo debemos agregar como se ve en pantalla en el `view_mode` el nuevo tipo calendar.

En esta modificación lo que hicimos fue agregar la vista calendar al `view_mode`, como ya explicamos anteriormente esto es porque Odoo utiliza este parámetro para determinar que vistas estarán disponibles en la interfaz de nuestro módulo.

Ahora que ya hemos implementado nuestra vista kanban en el `views.xml`, pero todavía nos falta hacer unas modificaciones en los bloques del formulario para poder agregar la fecha de inicio y fin a las tareas cuando las creemos o editemos, además de agregar estos nuevos campos de fecha a las vistas de lista y kanban:

```xml
<!-- ================================================
     VISTA DE FORMULARIO (form)
     ================================================
     Se usa al crear o editar una tarea.
     Más amigable y detallada que la vista de lista.
-->
<record id="view_lista_tareas_form" model="ir.ui.view">
  <field name="name">lista.tareas.form</field>
  <field name="model">lista_tareas.lista</field>
  <field name="arch" type="xml">
    <form string="Tarea">
      <!-- El contenedor visual principal del formulario -->
      <sheet>
        <!-- Primer bloque de campos (agrupados) -->
        <group>
          <field name="tarea"/>          <!-- Campo de nombre de tarea -->
          <field name="realizada"/>      <!-- Campo para marcar como realizada -->
        </group>

        <!-- Segundo bloque de campos (prioridad y urgencia) -->
        <group string="Prioridad y urgencia">
          <field name="prioridad"/>      <!-- Campo de prioridad editable -->

          <!-- Campo calculado: no editable. Ya no usamos attrs (Odoo 17+) -->
          <field name="urgente" readonly="1"/>
        </group>

        <!-- Tercer bloque de campos para las fechas de la tarea -->
        <group string="Fechas de la tarea">
          <field name="fecha_inicio"/>
          <field name="fecha_fin"/>
        </group>

      </sheet>
    </form>
  </field>
</record>
```

Figura 10: Actualizamos el bloque de formulario para tener un nuevo bloque para agregar las fechas de inicio y fin de la tarea para cuando agregemos una tarea o la editemos.

En el formulario, agregamos un nuevo grupo para las fechas de la tarea donde tenemos los campos de fecha de inicio y fecha final de la tarea, para cuando creemos y editemos una tarea podamos agregar las fechas.

```xml
<!-- ================================================
     VISTA DE LISTA (list)
     ================================================
     Muestra las tareas en forma de tabla.
     Es la vista que se usa por defecto para ver varios registros.
-->
<record id="view_lista_tareas_list" model="ir.ui.view">
  <!-- Nombre interno de la vista -->
  <field name="name">lista.tareas.list</field>
  <!-- Modelo al que pertenece esta vista -->
  <field name="model">lista_tareas.lista</field>
  <!-- Estructura XML de la vista -->
  <field name="arch" type="xml">
    <!-- Vista tipo list (reemplaza a <tree> en Odoo 17+) -->
    <list string="Tareas" default_order="prioridad desc">
      <!-- Campos que se muestran como columnas -->
      <field name="tarea"/>          <!-- Nombre de la tarea -->
      <field name="prioridad"/>      <!-- Nivel de prioridad -->
      <field name="urgente"/>        <!-- Calculado automáticamente -->
      <field name="realizada"/>      <!-- Estado de finalización -->
      <field name="fecha_inicio"/>   <!-- Fecha de inicio -->
      <field name="fecha_fin"/>      <!-- Fecha de fin -->
    </list>
  </field>
</record>
```

Figura 11: Actualizamos el bloque de la vista list para mostrar los nuevos campos de fechas de inicio y fin de la tarea.

En la vista de lista, agregamos los nuevos campos de fecha de inicio y fecha de final de la tarea como nuevas columnas.

```xml
<!-- ================================================
     VISTA DE KANBAN
     ================================================ -->
<record id="view_lista_tareas_kanban" model="ir.ui.view">
  <!-- Nombre interno de la vista -->
  <field name="name">lista.tareas.kanban</field>
  <!-- Modelo al que pertenece esta vista -->
  <field name="model">lista_tareas.lista</field>
  <!-- Estructura XML de la vista -->
  <field name="arch" type="xml">
    <!-- Vista tipo kanban -->
    <kanban default_order_by="realizada">
      <field name="tarea"/>
      <field name="prioridad"/>
      <field name="urgente"/>
      <field name="realizada"/>

      <templates>
        <!-- Tarjeta de las tareas de kanban -->
        <t t-name="kanban-box">
          <div class="oe_kanban_card oe_kanban_global_click">
            <strong><field name="tarea"/></strong>
            <div>
              <div>Prioridad: <field name="prioridad"/></div>
              <div>Urgente: <field name="urgente"/></div>
              <div>Realizada: <field name="realizada"/></div>
              <div>Fecha de inicio: <field name="fecha_inicio"/></div>
              <div>Fecha de fin: <field name="fecha_fin"/></div>
            </div>
          </div>
        </t>
      </templates>
    </kanban>
  </field>
</record>
```

Figura 12: Actualizamos el bloque de la vista kanban para mostrar los nuevos campos de fechas de inicio y fin de la tarea.

En la vista de kanban, agregamos los nuevos campos de fecha de inicio y fecha de final de la tarea en la tarjeta del kanban.

2.5. Comprobación de la vista Calendar

A continuación vamos a enseñar a través de capturas de pantalla el correcto funcionamiento de la vista calendar que acabamos de implementar.

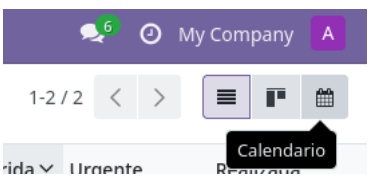


Figura 13: Como se pueder ver en esta captura para poder cambiar de vista nos iremos a estos iconos y pulsaremos en el que pone calendario.

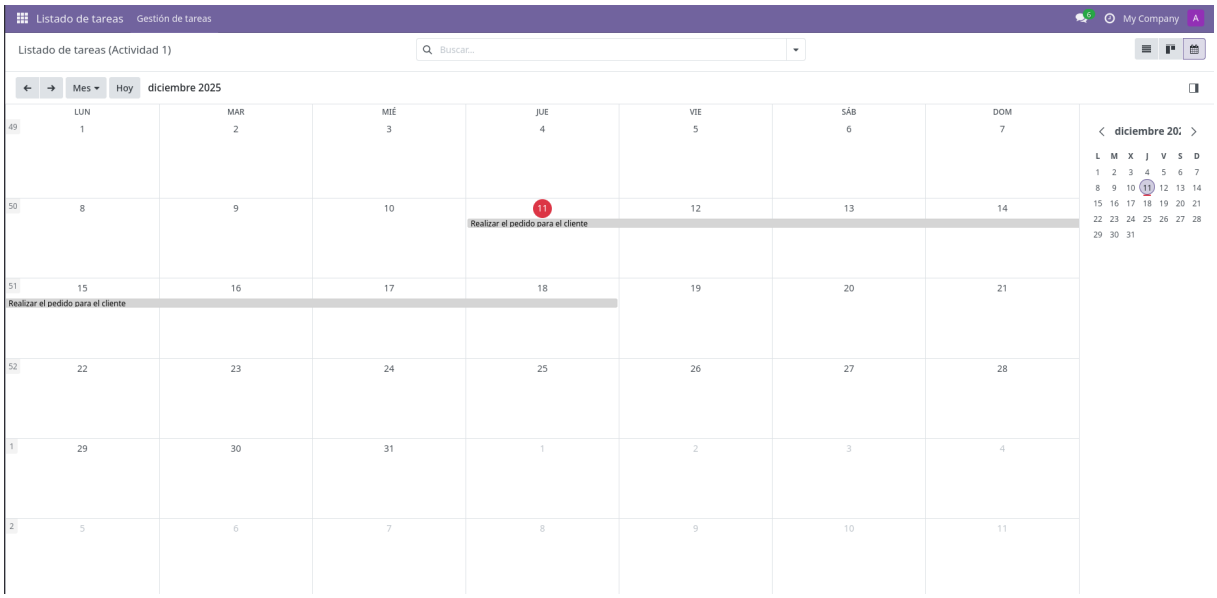


Figura 14: En esta captura podemos ver la vista modo calendar de nuestro módulo de listas de tareas.

Aquí podemos ver como la tarea que creamos de **realizar un pedido para un cliente**, empieza el jueves 4 y termina el viernes 18 de diciembre de 2025, lo que muestra que tenemos la vista del calendario bien implementada.

3. Actividad 02 - Biblioteca de cómics

3.1. Objetivo

Para esta actividad se nos pide ampliar el módulo del ejemplo 3 de la biblioteca de cómics simple para agregar la gestión de socios y la gestión de ejemplares de préstamo. Para ello crearemos nuevos modelos que nos permiten almacenar los datos de los socios y de los ejemplares, así como poder registrar los préstamos indicando el socio, la fecha de inicio y la fecha de devolución, aplicando las restricciones indicadas a las fechas.

3.2. Modelo Socios

A continuación vamos a crear el modelo de socios para nuesto módulo, para ello tendremos que crear la vista y el modelo además de realizar unas modificaciones para emplearlo.

```python
# -*- coding: utf-8 -*-
from odoo import models, fields

# Definimos el modelo Biblioteca socio
class BibliotecaSocio(models.Model):
    # Nombre del modelo
    _name = "biblioteca.socio"
    # Descripcion del modelo
    _description = "Socio de la bilioteca"
    # Usamos el id_socio como nobre del socio
    _rec_name = "id_socio"

    # Atributos del modelo: identificador, nombre y apellido
    id_socio = fields.Char("ID socio", required=True, index=True)
    nombre = fields.Char("Nombre", required=True)
    apellido = fields.Char("Apellido", required=True)

    #Constraints de SQL del modelo: el identificador deber ser único
    _sql_constraints = [
        ("identificador_uniq", "UNIQUE(identificador)", "El identificador debe ser único.")
    ]
```

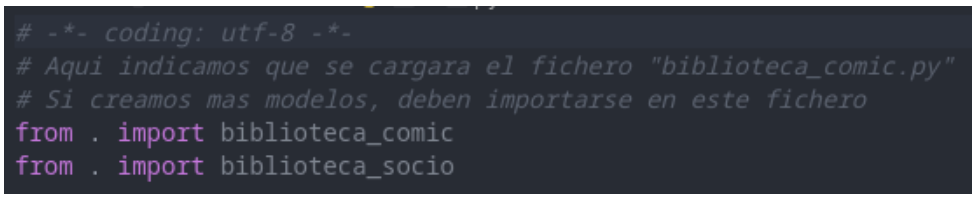
Figura 15: Captura del archivo `biblioteca_socio.py` donde tenemos el modelo para gestionar socios.

Creamos el archivo `biblioteca_socio.py` y definimos el modelo "biblioteca.socio", que se encargará de gestionar los socios de la biblioteca de cómics. Cada socio tiene un identificador único, además de su nombre y apellido.

Los atributos principales del modelo son:

- id_socio: identificador único para cada socio.
- nombre: nombre del socio.
- apellido: apellido del socio.

Además, usamos una restricción SQL para asegurar que el `id_socio` sea único.



```python
# -*- coding: utf-8 -*-
# Aqui indicamos que se cargara el fichero "biblioteca_comic.py"
# Si creamos mas modelos, deben importarse en este fichero
from . import biblioteca_comic
from . import biblioteca_socio
```

Figura 16: Captura de que importamos el nuevo modelo socio en `models/__init__.py`.

Para que Odoo cargue nuestro modelo de socios al iniciar el módulo, importamos el modelo en el archivo `models/__init__.py` como ya teniamos el del modelo de cómic.

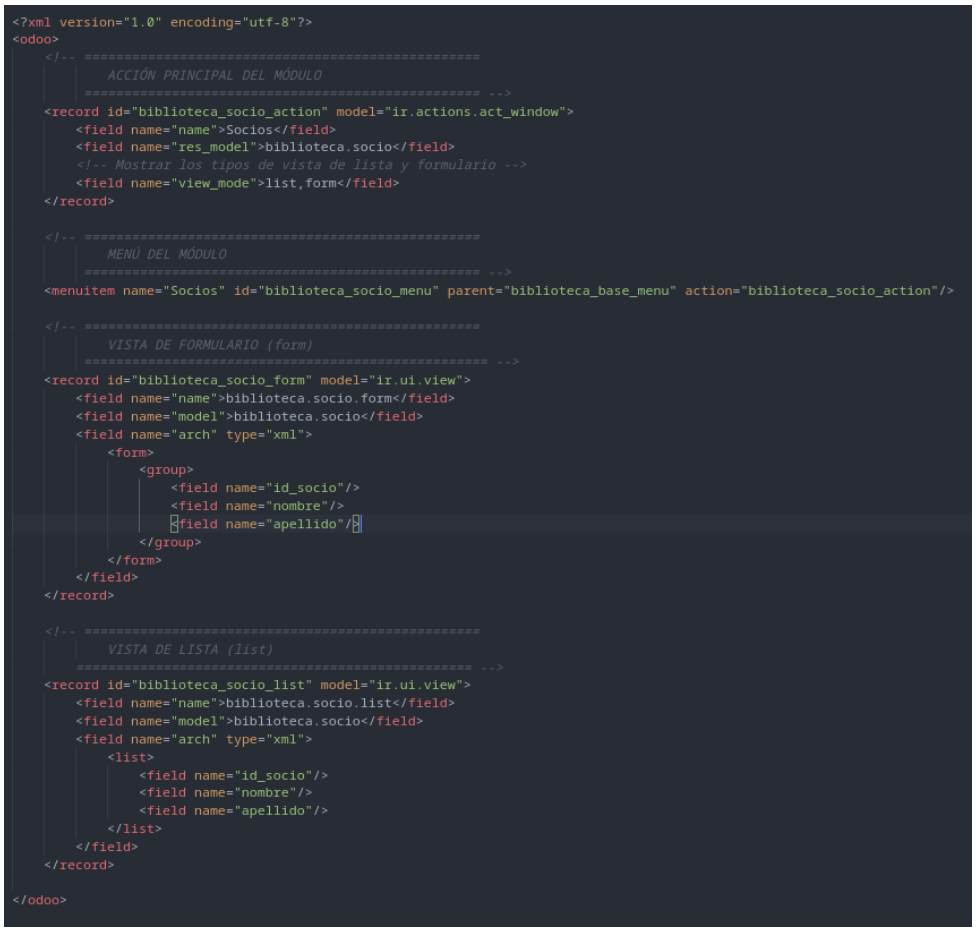


Figura 17: Captura del archivo `biblioteca_socio.xml` donde tenemos definida la vista para socios.

Creamos el archivo `biblioteca_socio.xml` donde creamos la vista `socio` para poder gestionar los socios desde una interfaz donde tenemos lo siguiente:

- **Vista formulario:** Esto nos permite tener un formulario donde podremos crear o editar la información de los socios.
- **Vista lista:** Esto nos muestra tados los socios en un listado formato tabla con los campos: identificador, nombre y apellido.

También agregamos la acción y el menú correspondientes para poder acceder al listado de socios desde la interfaz.

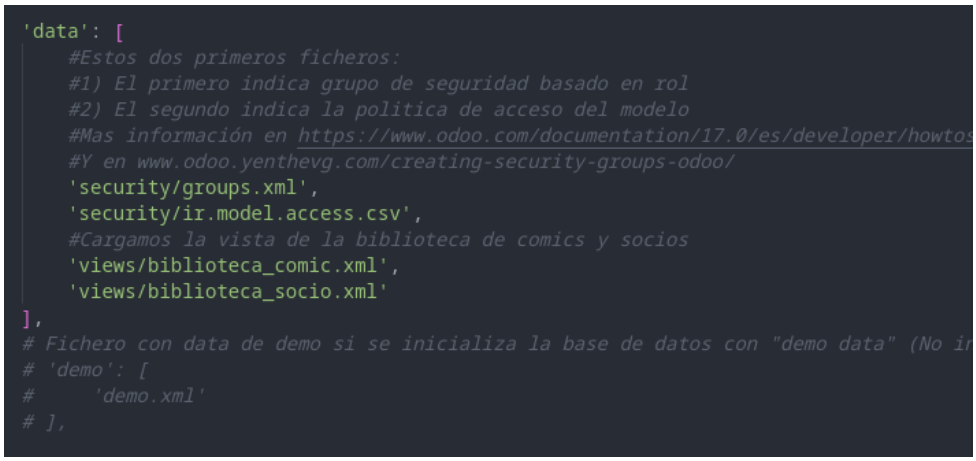


Figura 18: Captura de que incluimos la nueva vista de socio en `__manifest__.py`.

Para que Odoo cargue nuestro vista de socios, tenemos que agregarlo en `data` en el archivo `__manifest__.py` justo debajo de la vista de cómics.

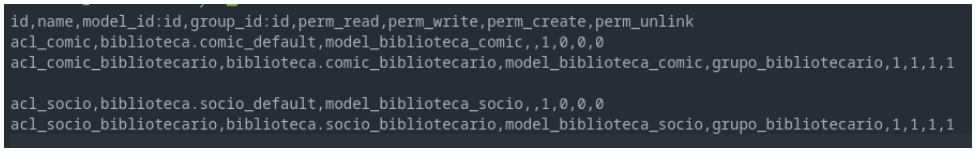
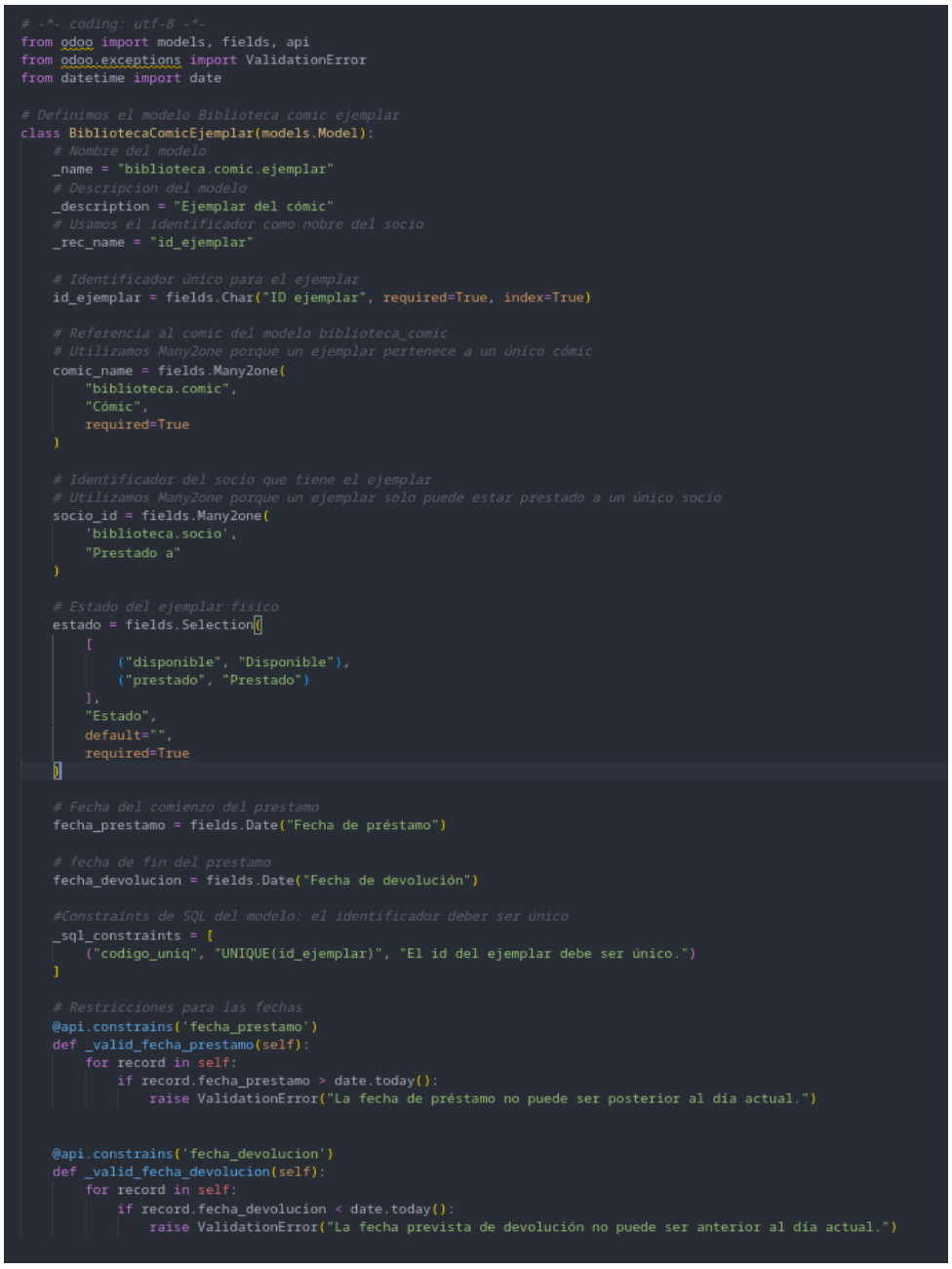


Figura 19: Captura donde actualizamos las reglas de acceso donde asignamos los permisos para socios.

Ahora, definimos las reglas de acceso, permitiendo a todos los usuarios poder ver los socios, pero solo los miembros del **grupo bibliotecario** podrán crear, editar y eliminar los socios.

3.3. Modelo Ejemplares

A continuación vamos a crear el modelo de ejemplares para nuesto módulo, para ello tendremos que crear la vista y el modelo además de realizar unas modificaciones para poder emplearlo.



```python
# -*- coding: utf-8 -*-
from odoo import models, fields, api
from odoo.exceptions import ValidationError
from datetime import date

# Definimos el modelo Biblioteca comic ejemplar
class BibliotecaComicEjemplar(models.Model):
    # Nombre del modelo
    _name = "biblioteca.comic.ejemplar"
    # Descripcion del modelo
    _description = "Ejemplar del cómic"
    # Usamos el identificador como nobre del socio
    _rec_name = "id_ejemplar"

    # Identificador único para el ejemplar
    id_ejemplar = fields.Char("ID ejemplar", required=True, index=True)

    # Referencia al comic del modelo biblioteca_comic
    # Utilizamos Many2one porque un ejemplar pertenece a un único cómic
    comic_name = fields.Many2one(
        "biblioteca.comic",
        "Cómic",
        required=True
    )

    # Identificador del socio que tiene el ejemplar
    # Utilizamos Many2one porque un ejemplar solo puede estar prestado a un único socio
    socio_id = fields.Many2one(
        'biblioteca.socio',
        "Prestado a"
    )

    # Estado del ejemplar físico
    estado = fields.Selection(
        [
            ("disponible", "Disponible"),
            ("prestado", "Prestado")
        ],
        "Estado",
        default="",
        required=True
    )

    # Fecha del comienzo del prestamo
    fecha_prestamo = fields.Date("Fecha de préstamo")

    # fecha de fin del prestamo
    fecha_devolucion = fields.Date("Fecha de devolución")

    #Constraints de SQL del modelo: el identificador deber ser único
    _sql_constraints = [
        ("codigo_uniq", "UNIQUE(id_ejemplar)", "El id del ejemplar debe ser único.")
    ]

    # Restricciones para las fechas
    @api.constrains('fecha_prestamo')
    def _valid_fecha_prestamo(self):
        for record in self:
            if record.fecha_prestamo > date.today():
                raise ValidationError("La fecha de préstamo no puede ser posterior al día actual.")


    @api.constrains('fecha_devolucion')
    def _valid_fecha_devolucion(self):
        for record in self:
            if record.fecha_devolucion < date.today():
                raise ValidationError("La fecha prevista de devolución no puede ser anterior al día actual.")
```

Figura 20: Captura del archivo `biblioteca_ejemplares.py` donde tenemos el modelo para gestionar los ejemplares.

Creamos el archivo `biblioteca_ejemplar.py` y definimos el modelo `"biblioteca.comic.ejemplar"`. Este modelo nos permite gestionar cada ejemplar de un comic de nuestra biblioteca. Cada ejemplar tiene un identificador único, una referencia al cómic al que pertenece, un estado y opcionalmente, un socio al que esta prestado y las fechas de prestamo y devolución.

Los atributos principales del modelo son:

- id_ejemplar: identificador único para cada ejemplar.
- comic_name: tenemos una relación `Many2one` hacia el modelo de cómic, ya que cada ejemplar pertenece a un único cómic.

- socio_id: tenemos una relación `Many2one` hacia el modelo de socio, ya que cada ejemplar esta prestado a un socio.
- estado: indica si el ejemplar está disponible o está prestado.
- fecha_prestamo: fecha en la que se presto el ejemplar.
- fecha_devolucion: fecha prevista en la que se devolvera el ejemplar.

Además, usamos una restricción SQL para asegurar que el `id_ejemplar` sea único.

Agregamos las restricciones para las fechas:

Incluimos dos comprobaciones mediante `@api.constrains`, así si se intenta guardar un ejemplar que incumpla alguna de estás reglas que nos salte un error:

1. Que la fecha de préstamo no puede ser posterior al día actual.
2. Que la fecha de devolución no puede ser anterior al día actual.

```
# -*- coding: utf-8 -*-
# Aqui indicamos que se cargara el fichero "biblioteca_comic.py"
# Si creamos mas modelos, deben importarse en este fichero
from . import biblioteca_comic
from . import biblioteca_socio
from . import biblioteca_ejemplar
```

Figura 21: Captura de que importamos el nuevo modelo ejemplar en `models/__init__.py`.

Para que Odoo cargue nuestro modelo de ejemplar al iniciar el módulo, importamos el modelo en el archivo `models/__init__.py` como ya teniamos el del modelo de cómic y socio.

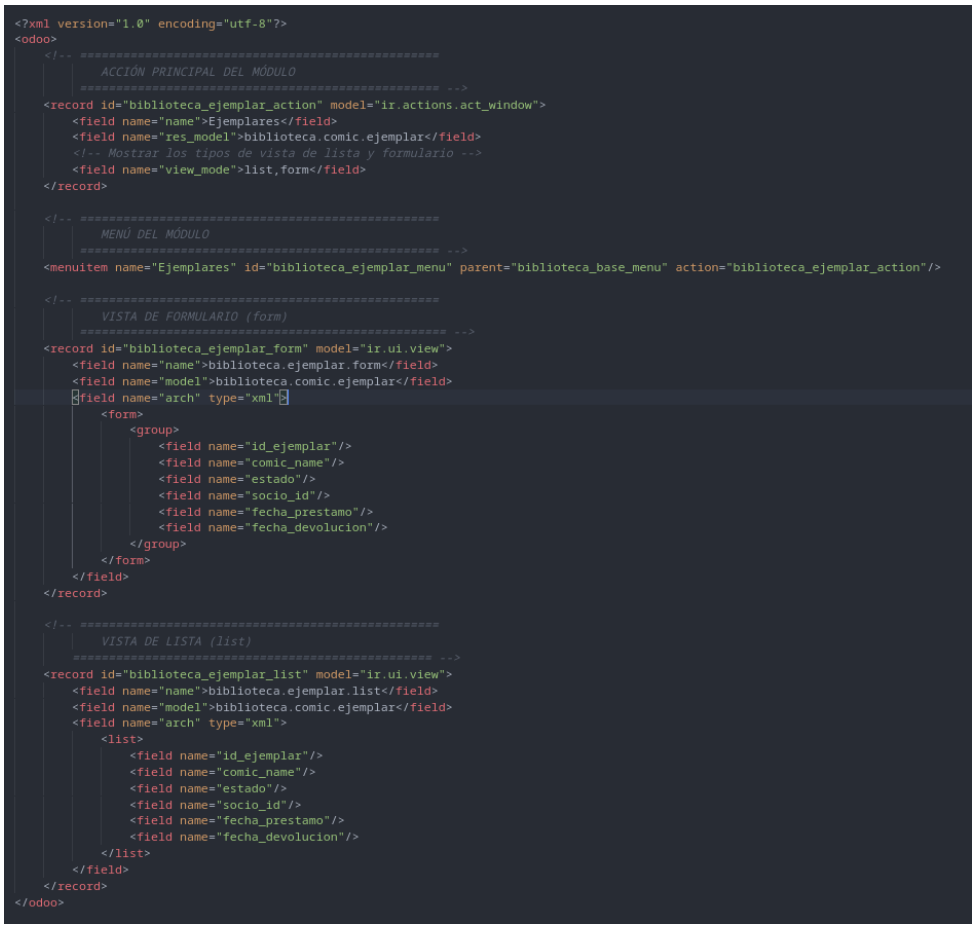


Figura 22: Captura del archivo `biblioteca_ejemplar.xml` donde tenemos definida la vista para los ejemplares.

Creamos el archivo `biblioteca_ejemplar.xml` donde creamos la vista `ejemplar` para poder gestionar los ejemplares desde una interfaz donde tenemos lo siguiente:

- **Vista formulario:** Esto nos permite tener un formulario donde podremos crear o editar la información de los ejemplares.
- **Vista lista:** Esto nos muestra tados los ejemplares en un listado formato tabla con los campos: identificador del ejemplar, nombre del cómic, estado, identificador del socio, fecha del prestamo y de devolución.

También agregamos la acción y el menú correspondientes para poder acceder al listado de ejemplares desde la interfaz.

```
'data': [
    #Estos dos primeros ficheros:
    #1) El primero indica grupo de seguridad basado en rol
    #2) El segundo indica la politica de acceso del modelo
    #Mas información en https://www.odoo.com/documentation/17.0/es/developer/howtos/rdtraining/05_securityintro.html
    #Y en www.odoo.yenthevg.com/creating-security-groups-odoo/
    'security/groups.xml',
    'security/ir.model.access.csv',
    #Cargamos la vista de la biblioteca de comics y socios
    'views/biblioteca_comic.xml',
    'views/biblioteca_socio.xml',
    'views/biblioteca_ejemplar.xml'
],
# Fichero con data de demo si se inicializa la base de datos con "demo data" (No incluido en ejemplo)
# 'demo': [
#     'demo.xml'
# ],
```

Figura 23: Captura de que incluimos la nueva vista de ejemplar en `__manifest__.py`.

Para que Odoo cargue nuestro vista de ejemplares, tenemos que agregarlo en `data` en el archivo `__manifest__.py` justo debajo de la vista de socios.

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
acl_comic,biblioteca.comic_default,model_biblioteca_comic,,1,0,0,0
acl_comic_bibliotecario,biblioteca.comic_bibliotecario,model_biblioteca_comic,grupo_bibliotecario,1,1,1,1

acl_socio,biblioteca.socio_default,model_biblioteca_socio,,1,0,0,0
acl_socio_bibliotecario,biblioteca.socio_bibliotecario,model_biblioteca_socio,grupo_bibliotecario,1,1,1,1

acl_ejemplar,biblioteca.ejemplar_default,model_biblioteca_comic_ejemplar,,1,0,0,0
acl_ejemplar_bibliotecario,biblioteca.ejemplar_bibliotecario,model_biblioteca_comic_ejemplar,grupo_bibliotecario,1,1,1,1
```

Figura 24: Captura donde actualizamos las reglas de acceso donde asignamos los permisos para ejemplares.

Ahora, definimos las reglas de acceso, permitiendo a todos los usuarios poder ver los ejemplares, pero solo los miembros del **grupo bibliotecario** podrán crear, editar y eliminar los ejemplares.

3.4. Comprobación de su funcionamiento

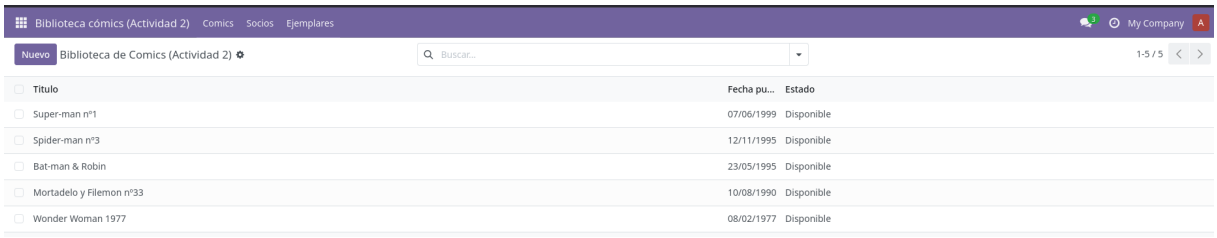


Figura 25: En esta captura tenemos la lista con todos los cómics que creamos para nuestra biblioteca.

Como podemos ver tenemos creados cinco cómics y vemos que se muestra correctamente en el formato lista, ademas podemos ver en el menú las opciones cómics, socios y ejemplares, para movernos de modelo.

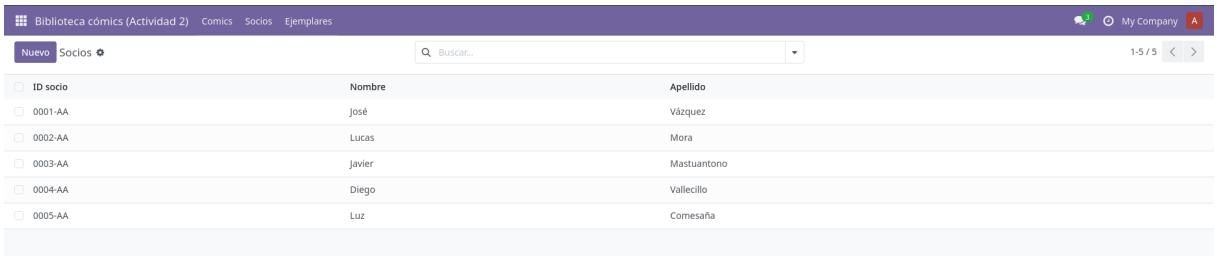


Figura 26: En esta captura tenemos la lista con todos los socios que hemos creamos para nuestra biblioteca.

Como podemos ver estamos en la parte del modulo de socios donde los gestionamos, creamos cinco socios de ejemplos y los mostramos en el formato lista correctamente.

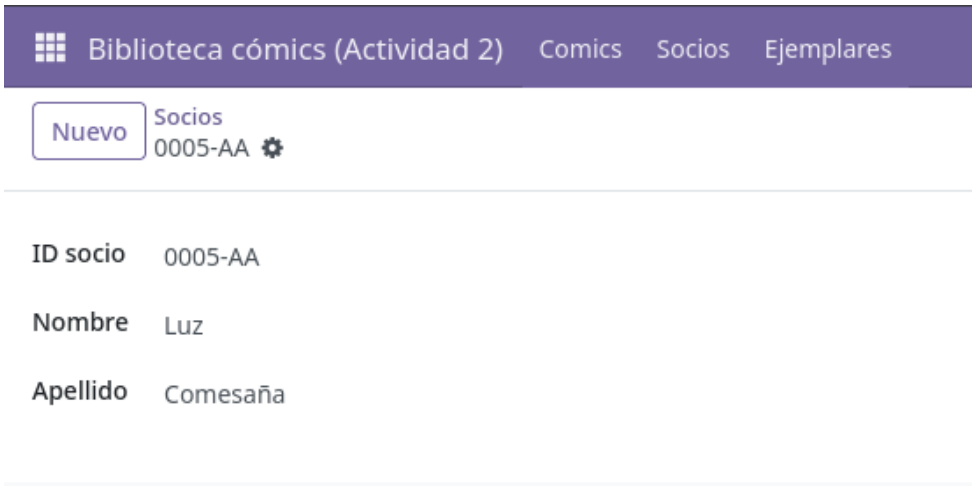


Figura 27: En está captura tenemos el formulario para crear y editar los socios.

Como podemos ver ahora estamos en la pantalla de crear o editar un socio donde podemos ver que tenemos los campos: identificador, nombre y apellido.

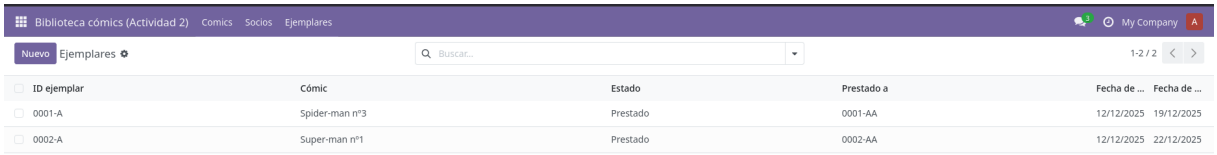


Figura 28: En esta captura tenemos la lista con todos los ejemplares que hemos creamos para nuestra biblioteca.

Ahora nos encontramos en la parte del modulo de ejemplares donde los gestionamos, he creado 2 ejemplares de ejemplos y los mostramos en el formato lista correctamente.

Figura 29: En está captura tenemos el formulario para crear y editar nuestros ejemplares.

Como podemos ver ahora estamos en la pantalla de crear o editar un ejemplar donde podemos ver que tenemos los campos: identificador del ejemplar, nombre del cómic, estado, a que socio está prestado, en que fecha se presto y cuando se va a devolver.

Por último vamos a mostrar las comprobaciones de que nuestras restricciones en las fechas en ejemplares están funcionando correctamente.

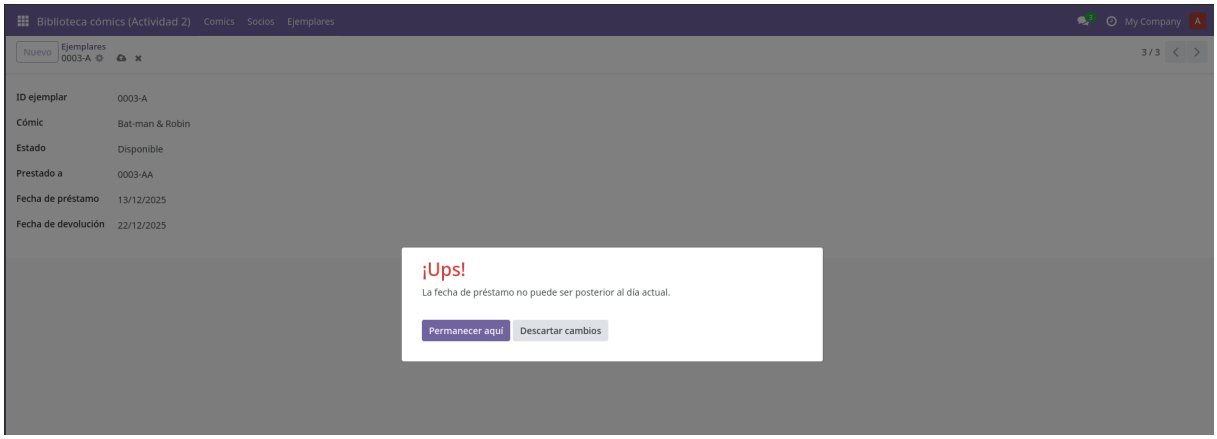


Figura 30: En está captura comprobamos que la fecha de préstamo no puede ser posterior al día actual.

Aquí podemos ver como al intentar guardar un nuevo ejemplar con la fecha de prestamo 13/12/2025 nos salta el error ya que el dia en la intentamos registrar el ejemplar era 12/12/2025 y como definimos en la restricción no podemos registrar un inicio de prestamo en un fecha futura.

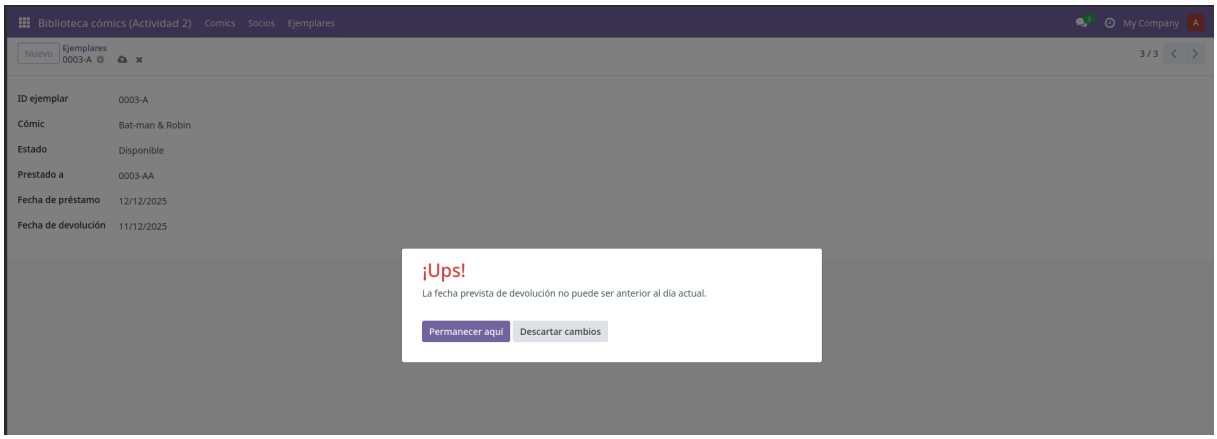


Figura 31: En está captura comprobamos que la fecha prevista de devolución no puede ser anterior al día actual.

Aquí podemos ver como al intentar guardar un nuevo ejemplar con la fecha de devolución 11/12/2025 nos salta el error ya que el dia en la intentamos registrar el ejemplar era 12/12/2025 y como definimos en la restricción no podemos registrar una devolución de prestamo en un fecha ya pasada al día actual.

4. Actividad 03 - Pacientes y médicos

5. Actividad 04 - Ciclos formativos

6. Conclusión